UNIVERSIDAD AUTÓNOMA METROPOLITANA
Unidad Iztapalapa

Examen #1

Olvera Monroy Gonzalo

<2173011224>

Sistemas Operativos

Prof. Orlando Muñoz Texzocotetla

Trimestre: 20 - O

Desarrollo:

1. Lo primero que hice fue estudiar el código que nos dio como base el profesor para realizar el examen.
2. Comencé a codificar una vez que entendí el código.
3. Quite la variable TAMNIO_BUFFER ya que el usuario tiene que dar el tamaño, entonces declare una variable global “n”.
4. Luego modifique a las clases de productor y consumidor, ya que pide que el usuario de los valores de cuantos productores y consumidores se van a crear. Para eso utilice **fork()**.
5. Por ultimo agregue en las clases productor y consumidor la variable **k** ya que pide que el usuario de ese valor para que cada productor y consumidor, produzca/consuma.

Monitor.

```
/*
      ************************
       Olvera Monroy Gonzalo
             Examen #1
      ************************
*/

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <unistd.h>

/*
Claves para semaforos. Estos valores son INDICES.
0 LLENO: Valor inicial será 0. Cuantos lugares estan ocupados en el buffer
1 VACIO:  Valor inicial el tamaño de buffer. Indica cuantos lugares hay disponibles para
consumir
2 MUTEX: Valor inicial 1. Este da permiso o bloquea el acceso a la región critica
*/
#define LLENO 0
#define VACIO 1
#define MUTEX 2

/*Es el tamanio de Memoria Compartida y del buffer que va a decir el usuario
La declaro como una variable global porque se va ocupar
en los metodos inicializa_valores, imprimir y borrar
*/
int n;


void inicializa_valores();
void imprimir();
void borrar();

int main(){
       int opcion;
```

```
	printf("\nDame el valor de n para el tamanio de la memoria compartida:");
	scanf("%d", &n);

	while(1){
		printf("\n\n1. Inicializar\n2. Imprimir bufer\n3. Borrar\n4. Salir");
		printf("\nIntroduce opcion: ");
		scanf("%d",&opcion);

		if(opcion == 1)
			inicializa_valores();
		else if(opcion == 2)
			imprimir();
		else if(opcion == 3)
			borrar();
		else if(opcion == 4)
			exit(0);
	}
}

/*
Se crea la variable compartida y se inicializa en cero
Se crean los semaforos y se inicializan sus valores
*/
void inicializa_valores(){
	int llave;
	int sem_id, shm_id, *buffer;
	ushort semaforos[3];

	// Creamos la llave
	llave = ftok(".", 'G');

	if(llave == -1)
		perror("\nError al general la llave\n");
	else{
		// Inicializamos los semáforo
		sem_id = semget(llave, 3, IPC_CREAT | 0666); // Se crean los semáforos
		if( sem_id == -1)
			perror("\nError al crear los semaforos\n");
		else{
			semaforos[LLENO] = 0;  // Espacios ocupados inicialmente en el bufer
			semaforos[VACIO] = n;  // Espacios inicialmente disponibles en el
buffer proporcionada por el usuario
			semaforos[MUTEX] = 1;   // Da o bloquea el acceso a la region critica
			semctl(sem_id, 0, SETALL, semaforos);   // Asigna a los semaforos los
valores en el arreglo semaforos
		}

		/*
		Inicializamos la variable compartida
		Con el tamanio que dio el usuario
		*/
		shm_id = shmget(llave, sizeof(int) * n, IPC_CREAT | 0666);
		if(shm_id == -1)
			perror("\nError al crear la variable compartida\n");
		else{
			buffer = (int *)shmat(shm_id, 0, 0); // Ligamos la memoria creada
anteriormente a este proceso
			*buffer = 0; // Valor inicial de la variable compartida
			shmdt((int *)buffer);  // Desligamos la memoria creada
		}
	} // Fin del if principal

}
```

```
// Borra los semaforos y la variable compartida
void borrar(){
	int llave, sem_id, shm_id;
	int *buffer;

	// Creamos la llave
	llave = ftok(".", 'G');

	if( llave == -1)
		perror("\nError al crear la llave\n");
	else{
		sem_id = semget(llave, 3, 0666);
		if(sem_id == -1)
			perror("\nError al encontrar semaforos\n");
		else{
			// Liberar semaforos
			if( (semctl(llave, 0, IPC_RMID)) < 0){
				perror("\nSe borro un espacio de memoria\n");
			}
		}

		// Borrando memoria
		shm_id = shmget(llave, sizeof(int) * n, 0666);
		if(shm_id == -1)
			perror("\nNo se encontro la memoria\n");
		else{
			shmctl(shm_id, IPC_RMID, NULL);
		}
	} // Fin del if principal
}

void imprimir(){
	int llave, shm_id, *buffer;
	int *array;
	// Creamos la llave
	llave = ftok(".", 'G');

	if(llave == -1)
		perror("\nNo se pudo crear la llave\n");
	else{
		shm_id = shmget(llave, sizeof(int) * n, 0666);
		if(shm_id == -1)
			perror("\nError al encontrar el buffer\n");
		else{
			int i = 0;
			buffer = (int *)shmat(shm_id, 0, 0); //Ligamos al bufer la memoria
compartida
			printf("\nEspacios vacios: %d", n- *buffer); // Muestra el valor del
buffer si esta vacio
			printf("\nEspacios llenos: %d\n", (*buffer)); // Muestra el valor del
buffer si hay un dato
			printf("\n");

			for(i = 0; i < n; i++) { // Muestra el dato insertado en cada casilla
por el productor {10}
				printf(" [%d] ", buffer[i]);
			}

			shmdt((char *)buffer);
		}
	}	// Fin del if principal
}
```

Para la implementación del monito se llevó a cabo de la siguiente manera:

1. Declare 1 variable global “n” que es para el tamaño de la memoria compartida y del buffer, ese valor lo proporciona el usuario.
2. Los métodos inicializa_valores(), borrar() e imprimir() nos lo proporciono el profesor, lo cual el método de inicializa_valores va a inicializar la variable compartida en 0 y a los semáforos. Al semáforo[VACIO] se le asigna el valor que dio el usuario para el tamaño del buffer que este “n”. Cuando se hace “shm_id= shmget(llave, sizeof(int) * n, 0666” el segundo parámetro que esta subrayado se le pasa el tamaño que dio el usuario ya que si no se le pasa la variable “n”, va a tener diferente tamaño y no va a corresponder con lo que dio el usuario.
3. El método de borrar va a borrar a los semáforos y a la variable compartida al igual que el método de inicializa “shm_id= shmget(llave, sizeof(int) * n, 0666” a la memoria compartida en el segundo parámetro le doy el tamaño para que se borre todo.
4. El método de imprimir pues ahí solo modifiqué un poco, va a mostrar tres mensajes, el primer mensaje muestra los espacios vacíos que tiene el buffer, el segundo mensaje muestra los espacios llenos que hay en el buffer y el tercer mensaje muestra el todo el arreglo con el valor que le di que fue “10”.
5. Solo que tuve un problema al momento de imprimir el arreglo y no pude solucionarlo, en la posición [0] me muestra el valor de los espacios llenos y a partir de la posición [1] ya me muestra el valor 10. El otro problema que ya no pude solucionar es que cuando consumo si se actualiza el mensaje de “Espacio vacío y Espacio lleno” pero cuando se imprime el arreglo no me quita el valor 10. Por ejemplo, tengo [10] [10] y consumo uno en teoría tiene que mostrarme [10] [0] pero no lo hace me sigue mostrando [10] [10], solo se me actualiza los dos primeros mensajes.

Productor.

```c
/*
      ************************
       Olvera Monroy Gonzalo
             Examen #1
      ************************
*/
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <unistd.h>

/*
Claves para semaforos. Estos valores son INDICES.
0 LLENO: Valor inicial será 0. Cuantos lugares estan ocupados en el buffer
1 VACIO:  Valor inicial el tamaño de buffer. Indica cuantos lugares hay disponibles para
consumir
2 MUTEX: Valor inicial 1. Este da permiso o bloquea el acceso a la región critica
*/
#define LLENO 0
#define VACIO 1
#define MUTEX 2

/*
Es el tamanio de productores que va a decir el usuario
para crear varios productores.
*/
int s;

/*
Es el tamanio que cada productor va a producir en el buffer
el valor lo va a decidir el usuario
*/
int k;

int contador =0; // Su funcion es que va a depositar un valor en cada casilla del buffer


void depositar_dato();
void waitS(int semaforo); // Recibe el indice del semaforo al que se le aplicara wait: -1
void signalS(int semaforo); // Recibe el indice del semaforo al que se le aplicara el
signal: +1


int main(){

       pid_t pid;

       printf("\nDame el valor de s para el tamanio de los productores: ");
       scanf("%d", &s);
       printf("\nDame el valor de k para producir: ");
       scanf("%d", &k);


       //Su funcion del for es verificar todos los productores que proporcione el usuario
       for(int i = 0; i < s; i++) {
              pid = fork();

              if(pid < 0){
```

```
                        perror("Fallo el fork");
                        exit(1);
                }

                /*
                Se verifica si el hijo es igual a 0
                Para crear a los "s" productores que dio el usuario
                */
                if( pid == 0){
                        sleep(1);
                        printf("\n\n\t     ____________________________________________\n");
                        printf("\n\t\tProceso %d, voy a producir %d elementos\n", i+1, k);

                        /*
                        Su funcion del for es que cada "s" productor que proporciono el
usuario
                        Cada uno va a producir "k" elementos
                        */
                        for(int j =0; j < k; j++) {
                                sleep(1);
                                printf("\n\t\t***********************************\n");
                                printf("\t\tP: Esperando por espacio disponible... \n");
                                waitS(VACIO);
                                sleep(1);
                                printf("\t\tP: Esperando acceso a la Region Critica... \n");
                                waitS(MUTEX);
                                contador ++; // Va a depositar en la siguiente casilla del
buffer
                                depositar_dato(); // Region Critica
                                sleep(1);
                                printf("\t\tP: Liberando acceso a buffer\n");
                                signalS(MUTEX);
                                sleep(1);
                                printf("\t\tP: Hay un elemento mas en buffer\n");
                                signalS(LLENO);
                                printf("\t\t***********************************\n");
                        }

                        printf("\t     ____________________________________________\n\n");

                } else {
                break;
                }
        }

}

// Es el equivalente al x++ del programa sin
// memoria compartida y sin semaforos de C
void depositar_dato(){
        int llave, shm_id, *buffer;;

        int item = 0; // Es el valor que se va a depositar en el buffer
        int j;

        // Creamos la llave
        llave = ftok(".", 'G');

        if( llave == -1 )
                perror("\nError al crear la llave\n");
        else{
                shm_id = shmget(llave, sizeof(int), 0666);
                if(shm_id == -1)
                        perror("\nError al encontrar la variable\n");
                else{
```

```
                        buffer = (int *)shmat(shm_id, 0, 0);
                        /*
                        Su funcion del for es que va a depositar el valor
                        */
                        for(j = 0; j < k; j++) {
                                sleep(1);

                                item = j + 10 ; // Va a depositar el valor "10" en el buffer
                                buffer[contador] = item; // Se va a depositar el valor de [10]
en cada casilla del arreglo
                                *buffer = *buffer + 1; // Se pasa a la siguiente casilla
                                printf("\t\tProductor produce {%d} en la posicion [%d]\n",
item, (*buffer -1));
                                break;
                        }
                        shmdt((char *) buffer);
                }
        }
}
```

Para la implementación del productor se llevó a cabo de la siguiente manera:

1. Declare 3 variables globales:
    a. La variable “s” es para cuantos productores van a crearse, ese valor lo proporciona el usuario.
    b. La variable “k” es para saber la cantidad que va a producir cada productor al igual es proporcionada por el usuario.
    c. La variable “contador” es para insertar un valor al buffer no lo proporciona el usuario.
2. Se crearon tres métodos que nos proporcionó el profesor los cuales son:
    a. El método depositar() se encargar de depositar un valor al buffer.
    b. El método es el waitS(int semaforo) como parámetro se le pasa el semáforo al que se le aplicara un wait.
    c. El método que es signalS(int semaforo) como parámetro se le pasa el semáforo al que se le aplicara un signal.
3. En el método de “depositar()” se le implemento lo siguiente:
    a. Se utilizaron 6 variables:
        i. La variable “llave” va a servir para dar una clave.
        ii. La variable “shm_id” va a servir para buscar la llave.
        iii. La variable “buffer” es un apuntador a la memoria.
        iv. La variable “item” es para depositar el valor al buffer y “j” es un contador.
4. Primero se va a ligar el buffer con la memoria, luego se hizo un ciclo “for” el cual se va a encargar de colocar el valor al buffer.
    a. Le asigno el valor a la variable “item” el valor de 10.
    b. Al buffer lo igualo a item para que se deposite el valor 10 en una casilla, la función del “contador” va a estar depositando el valor en la siguiente casilla del buffer.

c. Se aumenta el buffer para que se pase a la siguiente casilla, puse un mensaje para verificar que si se depositara el valor en cada una de las casillas del buffer.

5. En el main se pide los valores de “s” y “k”, una vez que el usuario da los valores.
   a. Se procede a crear los productores para eso hago un ciclo **for** para que se puedan crear los “s” productores que dio el usuario.
   b. Para crear varios productores es necesario utilizar **“fork()”**, se hace una verificación si el valor del pid es negativo entonces no se crea ningún productor, pero si es igual a 0 se empiezan a crear dentro del **if** que verifica si es igual a 0.
   c. Dentro del **if** hay un ciclo **for** el cual se encarga que cada productor produzca “k” elementos.
   d. Coloque unos mensajes para guiarme en que parte iba el código, antes de entrar a la región critica esta “contador++” se encarga de depositar el valor en la siguiente casilla del buffer.
   e. Coloque unos **sleeps** para que la pantalla del productor se viera más atractiva.

Consumidor.

```
/*
      ************************
       Olvera Monroy Gonzalo
             Examen #1
      ************************
*/

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <unistd.h>

/*
Claves para semaforos. Estos valores son INDICES.
0 LLENO: Valor inicial será 0. Cuantos lugares estan ocupados en el buffer
1 VACIO:  Valor inicial el tamaño de buffer. Indica cuantos lugares hay disponibles para
consumir
2 MUTEX: Valor inicial 1. Este da permiso o bloquea el acceso a la región critica
*/
#define LLENO 0
#define VACIO 1
#define MUTEX 2

/*
Es el tamanio de consumidores que va a proporcionar el usuario
para crear varios t consumidores.
*/
int t;
```

```
/*
Es el tamanio que cada consumidor va a consumir k elementos en el buffer
el valor lo va a decidir el usuario
*/
int k;

int contador = 0; // Su funcion es va a estar quitando el valor que hay en cada casilla del
buffer



void consumir();
void waitS(int semaforo); // Recibe el indice del semaforo al que se le aplicara wait: -1
void signalS(int semaforo); // Recibe el indice del semaforo al que se le aplicara el
signal: +1


int main(){
	pid_t pid;
	printf("\nDame el valor de t para el tamanio de los consumidores: ");
	scanf("%d", &t);

	printf("\nDame el valor de k para consumir: ");
	scanf("%d", &k);

	//Su funcion del for va a crear los consumidore que proporcione el usuario
	for(int i = 0; i < t; i++) {
		pid = fork();


		if(pid < 0) { //Se verificar que el valor del pid no sea negativo
			perror("Fallo el fork");
			exit(1);
		}

		/*
		Se verifica si el hijo es igual a 0
		Para crear a los "t" consumidores que dio el usuario
		*/
		if( pid == 0){
			sleep(1);
			printf("\n\n\t	_____\n");
			printf("\n\t\tProceso %d, voy a consumir %d elementos\n", i+1, k);

			/*
			Su funcion del for es que cada "t" consumidor que proporciono el
usuario
			Cada uno va a consumir "k" elementos
			*/
			for(int j =0; j < k; j++) {
				sleep(1);
				printf("\n\t\t**********************************\n");
				printf("\t\tC: Esperando por espacio disponible... \n");
				waitS(LLENO);
				sleep(1);
				printf("\t\tC: Esperando acceso a la Recion Critica... \n");
				waitS(MUTEX);
				contador++;
				consumir(); // Region Critica
				contador--; // Es quitar el valor que tiene el buffer
				sleep(1);
				printf("\t\tP: Liberando acceso a bufer\n");
				signalS(MUTEX);
				sleep(1);
				printf("\t\tC: Hay un elemento fuera del buffer\n");
```

```
                                signalS(VACIO);
                                printf("\t\t**********************************\n");

                        }
                        printf("\t      ____________________________________________\n\n");
                } else {
                        break;
                }
        }
}

// Es el equivalente al x++ del programa sin
// memoria compartida y sin semaforos de C
void consumir(){
        int llave, shm_id, *buffer;
        int item = 0; // Es el valor que se va a consumir en el buffer


        int j;

        // Creamos la llave
        llave = ftok(".", 'G');

        if( llave == -1 )
                perror("\nError al crear la llave\n");
        else{
                shm_id = shmget(llave, sizeof(int), 0666);
                if(shm_id == -1)
                        perror("\nError al encontrar la variable\n");
                else{
                        buffer = (int *)shmat(shm_id, 0, 0);

                        for(j = 0; j < k; j++){ // Se va a quitar el valor del buffer
                                sleep(1);
                                item = buffer[contador]; //Se quita el valor {10} que contiene
el buffer
                                *buffer = *buffer - 1; // Se retrocede una casilla
                                printf("\t\tConsumidor: consumo {%d} en la posicion: [%d]\n",
item, *buffer);
                                break;
                        }
                        shmdt((char *) buffer);
                }
        }

}
```

Para la implementación del consumidor se llevó a cabo de la siguiente manera:

1. Declare 3 variables globales:
    a. La variable “t” que es para cuantos consumidores van a crearse, ese valor lo proporciona el usuario.
    b. La variable “k” es para saber la cantidad que va a consumir cada consumidor al igual es proporcionada por el usuario.
    c. La variable “contador” es para quitar el valor que contiene el buffer no lo proporciona el usuario.
2. Se crearon tres métodos que nos proporcionó el profesor los cuales son:
    a. El método consumir() este método se encarga de consumir un valor al buffer.

b. El método waitS(int semaforo) como parámetro se le pasa el semáforo al que se le aplicara un wait.
c. El método que es signalS(int semaforo) como parámetro se le pasa el semáforo al que se le aplicara un signal.

3. El método de “consumir()” se le implemento lo siguiente:
    a. Se utilizaron 6 variables:
        i. La variable “llave” sirve para dar una clave.
        ii. La variable “shm_id” va a servir para buscar la llave.
        iii. La variable “buffer” es un apuntador para la memoria.
        iv. La variable “item” es el valor que tiene buffer y “j” es un contador.
4. Primero se va a ligar el buffer con la memoria, se hizo un ciclo “for” el cual se va a encargar de quitar el valor al buffer.
    a. El valor de variable “item = 10” ya que es que contiene el buffer, después se hizo una asignación “item= buffer[contador]” para estar quitando el valor 10 en las casillas.
    b. La función del “contador” es estar quitando el valor de la última casilla.
    c. Al buffer lo decremento para que se pase a la anterior casilla.
    d. Coloque un mensaje para verificar que si se consumiera el valor en cada una de las casillas del buffer.
5. En el main se pide los valores de “t” y “k”, una vez que el usuario da los valores.
    a. Se procede a crear los consumidores para eso hice un ciclo **for** para que se puedan crear los “t” consumidores que dio el usuario.
    b. Para crear varios consumidores es necesario utilizar **“fork()”**, se hace una verificación si el valor del pid es negativo entonces no se crea ningún consumidor, pero si es igual a 0 se empiezan a crear dentro del **if** que verifica si es igual a 0.
    c. Dentro del **if** hay un ciclo **for** el cual se encarga que cada consumidor consuma “k” elementos.
    d. Coloque unos mensajes para guiarme en que parte iba el código, despues de entrar a la región critica esta “contador--” se encarga de quitar el valor en el buffer.
    e. Coloque unos **sleeps** para que la pantalla del consumidor se viera más atractiva.

Cuestionario.

1. ¿Cuál es el comportamiento del sistema al tener más productores que consumidores?

Al momento de tener más productores que consumidores, va a depender del tamaño del buffer por ejemplo de tamaño 15, si son 5 productores y cada uno va a depositar 4 elementos al buffer, pues algún proceso estará en espera para entrar a la región critica ya que excede el valor del buffer, entonces se tiene a solo un consumidor que va a consumir 2 elementos en ese momento consume y el productor entra a la región critica pero vuelve a esperar para depositar el siguiente valor ya que no fue suficiente porque solo se tiene a un solo consumidor.

Eso sería el comportamiento del sistema cuando se tiene más productores que consumidores, el productor se quedara esperando para entrar a la región critica, ya que el número de productores es mayor a los consumidores y no es suficiente para que un solo consumidor libere espacio suficiente para que todos los productores no estén en espera a la región critica.

2. ¿Cuál es el comportamiento del sistema al tener más consumidores que productores?

Al momento de tener más consumidores que productores, va a depender del tamaño del buffer por ejemplo de tamaño 15, si son 5 consumidores y cada uno va a consumir 4 elementos al buffer, pues algún proceso estará en espera para entrar a la región critica ya que el buffer está vacío, entonces se tiene a solo un productor que va a producir 2 elementos en ese momento produce y el consumidor entra a la región critica pero vuelve a esperar para consumir el siguiente valor ya que no fue suficiente porque solo se tiene a un solo productor.

Eso sería el comportamiento del sistema cuando se tiene más consumidores que productores, el consumidor se quedara esperando para entrar a la región critica, ya que el número de consumidores es mayor a los productores y no es suficiente para que un solo productor pueda llenar el suficiente espacio para que todos los consumidores no estén en espera a la región critica.

3. Al aumentar el número de elementos que puede consumir o producir cada proceso ¿cuál es el comportamiento del sistema?

Si el usuario llega a dar un valor más grande que el tamaño del buffer para consumir o producir, algún proceso ya sea el consumidor o productor se quedara en espera para entrar a la región critica por motivos de que excedió el tamaño del buffer. Por ejemplo, para el caso de consumir:
El usuario quiere que el tamaño del buffer sea de 15, al principio el buffer esta vacío, en ese momento el usuario da el valor de 20 para que un proceso consuma los 20 elementos, pero el buffer este vacío entonces el proceso estará en espera para entrar a la región critica

porque no hay ningún elemento por consumir. Podrá entrar a la región critica el consumidor cuando un productor empiece a producir, pero si el productor solo produce 5 elementos, el consumidor seguirá en espera para entrar a la región critica ya que el valor de consumir es mayor al valor de producir.
Para el caso de producir el usuario da el valor de 20 para que un proceso produzca 20, pero si el buffer este vacío solo se podrá producir 15 elementos porque excedió el valor del buffer y quedara en espera para entrar a la región critica al menos que el consumidor empiece a consumir es cuando el podrá entrar a la región critica. Otro caso seria es que el buffer este lleno y el proceso empiece a producir 20 elementos, también estaría en espera ya que el buffer está lleno podrá entrar hasta que el consumidor empiece a consumir y el productor pueda entrar a la región crítica y así pueda depositar los 20 elementos en el buffer.

4. Al ejecutar el programa con diferentes valores para k, n, s, t ¿ocurre problemas de deadlock? explique por qué sí o por qué no.

A continuación, explico que es cada variable:

k: es el número de elementos que se va a producir/consumir lo proporciona el usuario.

n: es el valor del tamaño del arreglo, lo proporciona el usuario.

s: es el número de productores que se van a crear, lo proporciona el usuario.

t: es el número de consumidores que se van a crear, lo proporciona el usuario.

Deadlock.

En un entorno multiprogramación (donde dos o más procesos se pueden alojar en memoria y se ejecutados concurrentemente) muchos procesos compiten por una cantidad finita de recursos. Cuando un proceso requiere recursos y estos no están disponibles en ese momento entra a un estado de espera. Los procesos en espera pueden permanecer así indefinidamente si los recursos que solicitan están ocupados por otros procesos.
Con la anterior definición daré algunos ejemplos en donde se puede dar este fenómeno:

Ejemplo # 1:
En donde n < s = t y k toma el mismo valor para s y t
n=5, s,t = 6 y k=3
Para este caso el buffer (arreglo) es de tamaño 5, entonces hay 6 productores que cada uno va a producir 3 elementos teniendo un total de 18 elementos y solo el buffer es de tamaño 5 entonces algún productor va a tener que esperar hasta que el consumidor consuma, pero de todas maneras si el consumidor ya consumió todos los elementos, pero falta un proceso por consumir también quedara en espera a la región critica. Entonces significa que tenemos un deadlock.

Ejemplo #2:
En donde n > s = t y k toma el mismo valor para s y t.
n=10, s,t = 3, k=3
Para este caso el buffer es de tamaño 10, entonces hay 3 productores que cada uno va a producir 3 elementos teniendo un total de 9 elementos que va a depositar en el buffer, lo cual no tendrá problemas para depositar esos 9 elementos ya que el buffer le sobra una casilla entonces no tenemos un deadlock, al igual que el consumidor va a consumir esos 9 elementos y no va a tener problemas de esperar, en cambio sí aun no se le deposita nada al buffer y se empieza a consumir pues el consumidor entrara en un deadlock.

Ejemplo #3:
En donde n != s > t y k toma diferente valor para s y t
n=10, s=4 su valor de k=3, t=1 su valor de k=2
Para este caso el buffer es de tamaño de 10, entonces hay 4 productores cada uno va a producir 3 elementos teniendo un total de 12 elementos que se van a depositar en buffer, lo cual ya tendrá problemas porque el tamaño del buffer es inferior a la cantidad depositada entonces entrara en deadlock y como el consumidor es menor al productor no será suficiente para evitar que el productor entre a un deadlock.

Ejemplo #4:
En donde n!= t > s y k toma diferente valor para s y t
n=13, s=2 su valor de k=2, t=4 su valor de k=4
Para este caso el buffer es de tamaño 13, entonces solo hay 2 productores que cada uno va a producir 2 elementos teniendo un total de 4 elementos que se van a depositar en el buffer, entonces no tendrá problemas para depositar todos los elementos ya que tiene suficiente espacio para depositarlos entonces no entrara en un deadlock, el problema es que tenemos 4 consumidores que cada uno va a consumir 4 elementos teniendo un total de 16 elementos que se van a consumir en el buffer, entonces se van a consumir los 4 elementos que se depositaron pero algún proceso se quedara en espera ya que no hay más elementos a consumir porque el buffer está vacío, entonces entra en un deadlock.